Projet C:
Implémentation d'un *Ransomware*

Adrien Voisin, Didier Valentin, Bastien Bodart
Développement - IR209

Henallux - 2021-2022

"Connais ton ennemi et connais-toi toi-même eussiez-vous cent guerres à soutenir, cent fois vous serez victorieux"[1]

1 Introduction

Appréhender la sécurité informatique en se positionnant sous l'angle d'un attaquant est une technique assez répandue, particulièrement lors des tests d'intrusion avec la mise en place d'équipes dites de *Red Teaming*. Par contre, pour ce qui concerne les logiciels malveillants, on se limite généralement à renforcer la protection des systèmes et à terme, à mettre en quarantaine les programmes suspects sans pour autant se pencher sur le processus de développement de ceux-ci. Pourtant, les virus et autres malwares ne sont que des logiciels parmi d'autres, à l'exception près qu'ils ont pour objectif de mettre à mal un système informatique. Comme nous le verrons, il est important de désacraliser les virus informatiques et quoi de mieux pour cela que d'en réaliser un soi-même ? L'objectif de ce projet est de développer un *ransomware* ou rançongiciel en français, afin de mieux comprendre, à la fois le fonctionnement de ce type de programme, mais également de pouvoir appliquer les concepts vus au cours en utilisant le langage C pour réaliser le projet.

2 Fonctionnement d'un ransomware

Les *ransomware* sont une famille de logiciels malveillants dont l'objectif est, en général, de chiffrer des données sensibles sur un système afin d'obtenir une rançon en échange de la clé qui permettra de déchiffrer ces données. La maximisation des gains est donc primordiale pour l'attaquant et l'implémentation du *ransomware* sera impactée par cet objectif. WannyCry est un exemple typique de ce genre de logiciel[2]

[1] Sun Tzu, l'art de la guerre, VI siècle avant J-C
[2] https://fr.wikipedia.org/wiki/WannaCry

Figure 1: Ecran post-chiffrement du ransomware Wannacry.

Il existe de nombreux ransomware avec des fonctionnements cryptographiques et de propagation parfois fort différents. L'objectif de ce projet est de réaliser un ransomware relativement simple qui sera en mesure de chiffrer le contenu d'un répertoire. Le chemin du répertoire sera passé en paramètre et la clé de déchiffrement sera envoyée via le réseau. **L'infection de la machine par le malware ainsi que la propagation de celui-ci ne font pas partie des critères minimum de réussite**. On vous demande de vous limiter dans un premier temps au chiffrement des données.

3 Architecture du projet

Pour tester votre malware, il vous faudra deux machines virtuelles (de type Linux). Ces deux machines doivent pouvoir communiquer ensemble (vous pouvez, par exemple, configurer un réseau interne virtuel entre vos deux machines). Une fois le *ransomware* lancé sur la machine de la victime, une clé sera générée et utilisée pour chiffrer le contenu d'un répertoire. Une fois le chiffrement terminé, la clé sera envoyée à l'aide des *sockets* sur la machine de l'attaquant.

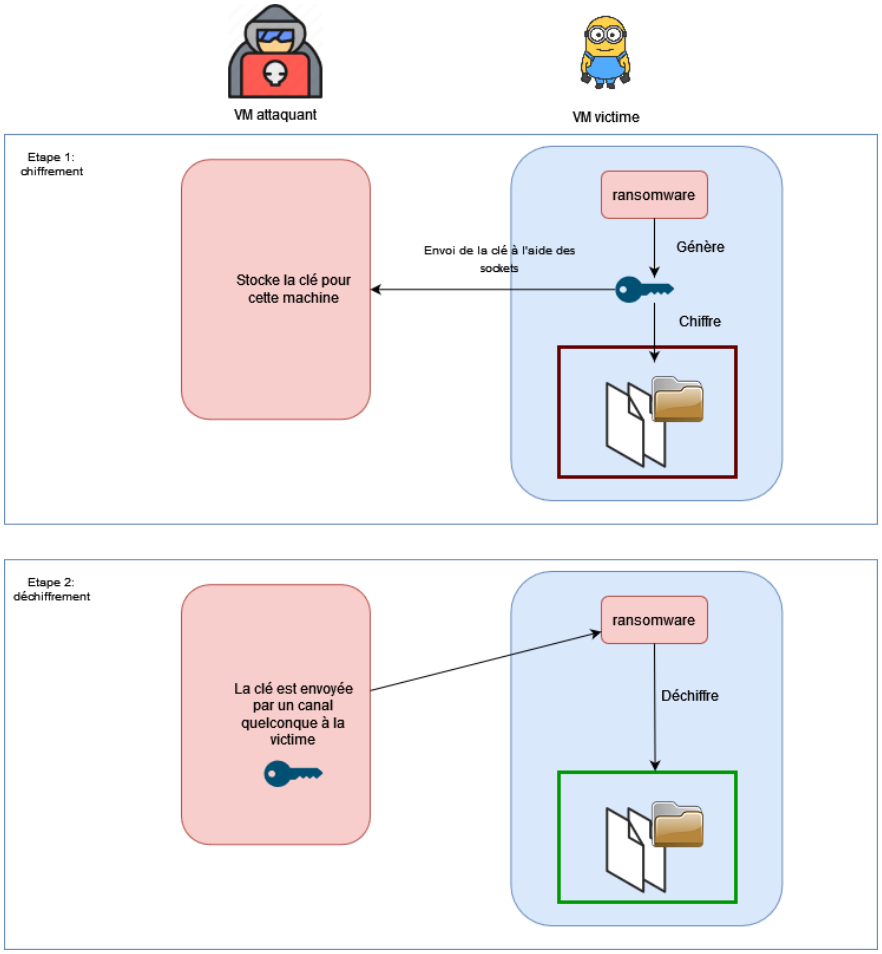

Figure 2: Architecture du projet.

4 Architecture du ransomware

Afin de vous aider dans l'implémentation du logiciel malveillant, une librairie de base vous est fournie : *ransomlib.h* et *ransomlib.c*. Cette librairie vous permettra de chiffrer et déchiffrer des fichiers. Il vous reste donc à implémenter le contenu de la fonction *ransom.c*.

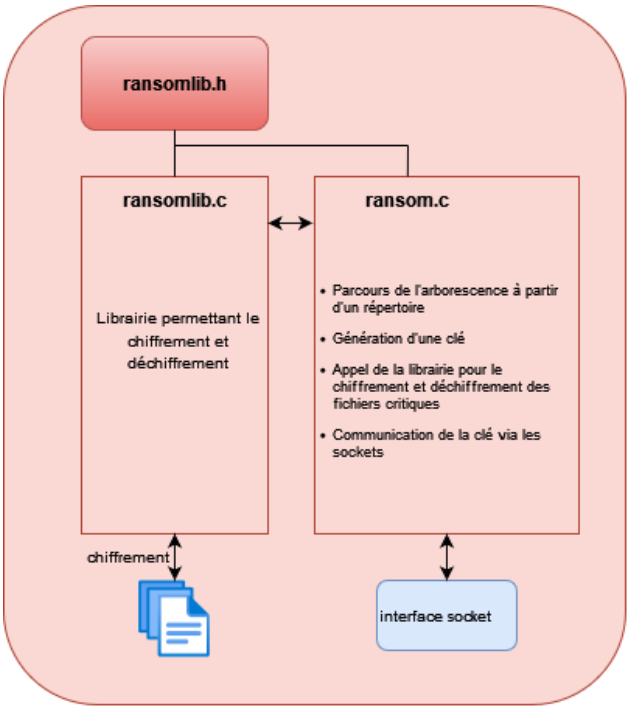

Figure 3: Architecture du ransomware.

4.1 Librairie ransomlib

Cette librairie repose sur une autre librairie (*openssl*) qui implémente toute une série d'algorithmes de chiffrement. *ransomlib* est actuellement composée des fonctions suivantes:

```
void handleErrors(void);

int bytes_to_hexa(const unsigned char bytes_string[],
    char *hex_string, int size);

void hexa_to_bytes(char hex_string[], unsigned char val[], int size);

int encrypt(unsigned char *key, unsigned char *iv, char* plaintext_file);

int decrypt(unsigned char *key, unsigned char *iv, char* cipher_file);
```

Deux fonctions sont utilisées pour convertir des chaines de caractères en binaire ou en hexadécimal. Ces conversions peuvent s'avérer utile pour faciliter le transfert et l'affichage des clés de chiffrement.

- *bytes_to_hexa*: permet de convertir une chaine de caractères binaire en chaine de caractères représentée en hexadécimal
- *hexa_to_bytes*: permet de convertir une chaine de caractères représentée en hexadécimal vers un format binaire

Les deux dernières fonctions permettent de respectivement chiffrer et déchiffrer des fichiers

- *encrypt*: prend comme paramètre une clé et un vecteur d'initialisation (en binaire) et le chemin d'un fichier. Il utilise les deux premiers paramètres pour chiffrer le fichier à l'aide de l'algorithme AES_265_CBC (voir section sur les aspects cryptographiques).
- *decrypt*: prend comme paramètre une clé et un vecteur d'initialisation (en binaire) et le chemin d'un fichier. Il utilise les deux premiers paramètres pour déchiffrer le fichier à l'aide de l'algorithme AES_265_CBC.

4.2 Programme principal (ransom.c)

Le programme principal a pour objectif de lister le contenu d'un répertoire et, en fonction des paramètres donnés à l'exécution, de soit chiffrer chaque fichier rencontré dans l'arborescence concernée, ou alors de les déchiffrer. Ce programme principal utilisera la librairie *ransomlib* pour les opérations de chiffrement et déchiffrement. C'est également ce programme principal qui sera en charge de générer une clé de chiffrement et de la communiquer à l'attaquant à l'aide des *sockets*. Les fonctions que vous allez devoir implémenter sont les suivantes:

```
void usage();

int is_encrypted(char *filename);

void list_dir(const char *name, unsigned char *iv,
    unsigned char *key, char de_flag);

int generate_key(unsigned char *key, int sizeKey,
    unsigned char *iv, int sizeIv,char *pKey, char *pIv);

int send_key(char *pKey, char *pIv);

int main (int argc, char * argv[]);
```

- usage: fonction permettant d'afficher l'aide
- is_encrypted: cette fonction permet de vérifier si un fichier est chiffré ou non. Cela permet par exemple d'éviter de chiffrer plusieurs fois le même fichier et de déchiffrer un fichier qui ne l'est pas. Attention qu'il ne s'agit pas de vérifier le contenu du fichier mais d'utiliser l'extension dans le nom du fichier.
- list_dir: fonction utilisée de manière récursive afin de lister tous les éléments présents dans un répertoire. Pour chaque fichier trouvé, cette fonction utilisera la librairie *ransomlib* pour les opérations de chiffrement en passant la clé et le vecteur d'initialisation en paramètres. Le dernièr paramètre *de_flag* permet lui d'indiquer si l'on souhaite chiffrer ou déchiffrer le contenu d'un répertoire. Ce *flag* devra être passé en argument lors de l'exécution du programme.

- generate_key: cette fonction a pour objectif de générer une clé et le vecteur d'initialisation correspondant. Cette fonction recoit comme paramètre l'adresse des couples clé:iv au format binaire (key, iv) et au format hexadécimal (pKey,pIv). Ces variables sont déclarées dans la fonction principale *main* et initialisées dans la fonction *generate_key* à l'aide de la librairie OpenSSL [3] pour le binaire et de la librairie ransom.h pour l'hexadécimal[4]. La liste complète des paramètres est définie ci-dessous:
  - key: pointeur vers la clé au format binaire déclarée dans la fonction *main*. Cette variable est un tableau de caractères non signés dont la taille est définie par une constante OpenSSL représant la taille d'une clé AES 256 bits.
  - sizeKey: taille d'une clé AES 256 bits. La fonction de génération aléatoire d'OpenSSL a besoin de cette information pour créer la clé.
  - iv: pointeur vers le vecteur d'initialisation au format binaire déclarée dans la fonction *main*. Cette variable est un tableau de caractères non signés dont la taille est définie par une constante OpenSSL représentant la taille d'un bloc CBC AES.
  - sizeIv: taille d'un bloc CBC AES. La fonction de génération aléatoire d'OpenSSL a besoin de cette information pour créer le vecteur d'initialisation.
  - pKey: pointeur vers un tableau de caractères déclaré dans la fonction *main*. Cette variable doit être initialisée en transformant la clé au format binaire vers un format hexadécimal. La taille de ce tableau doit être suffisante pour stocker une version hexadécimale de la clé binaire[5].
  - pIv: pointeur vers un tableau de caractères déclaré dans la fonction *main*. Cette variable doit être initialisée en transformant le vecteur d'initialisation au format binaire vers un format hexadécimal. La taille de ce tableau doit être suffisante pour stocker une version hexadécimale du vecteur binaire.
- send_key: on utilise cette fonction pour envoyer la clé et le vecteur d'initialisation via un *socket* vers la machine de l'attaquant.
- main: la fonction principale aura pour objectif de traiter les arguments passés à l'exécution (*argv*). On retrouvera dans les arguments au minimum: le chemin vers le répertoire à chiffrer, le mode d'opération (déchiffrer ou chiffrer) ainsi que la clé et le vecteur d'initialisation en cas de déchiffrement.

Vous pouvez choisir d'ajouter d'autres fonctions si nécessaire mais également d'adapter les signatures existantes si vous jugez cela pertinent. Attention toutefois de commencer par faire fonctionner votre *malware* en respectant la structure proposée.

Pour compiler et lancer le squelette de code fourni, vous pouvez taper les commandes suivantes:

[3] https://www.openssl.org/docs/man1.1.1/man3/RAND_bytes.html

[4] voir fonction bytes_to_hexa

[5] Il faut généralement 2x la taille d'un tableau binaire pour stocker sa version hexadécimale. Attention également de ne pas oublier un emplacement pour le caractère vide.

```
// installation librairie OpenSSL si nécessaire
sudo apt-get install libssl-dev
// faire le lien avec la librairie OpenSSL
gcc -o ransom ransom.c ransomlib.c -lcrypto
```

5 Cryptographie et OpenSSL

5.1 Critères cryptographiques du ransomware

Lors du travail en autonomie, il vous a été demandé de réfléchir au fonctionnement d'un *ransomware* et plus particulièrement aux aspects cryptographiques de celui-ci. En effet, un cybercriminel va tenter de trouver une manière de chiffrer et de déchiffrer des données sensibles d'une manière à maximiser ses profits. Voici une liste non exhaustive des critères intéressants du point de vue de cet attaquant:

- Chiffrement rapide : l'attaque doit être la plus rapide possible afin d'éviter la détection ou la mise hors tension des machines (dans le cas d'une propagation par le réseau par exemple)
- Clé unique par victime: pour maximiser ses profits, l'attaquant va favoriser l'utilisation d'une clé de déchiffrement unique par victime (en général, une clé par organisation, afin de faciliter l'opération de déchiffrement), afin d'empêcher cette victime de partager la clé de déchiffrement à d'autres organisations également touchées par ce logiciel malveillant.
- Utilisation d'un algorithme suffisamment fort pour éviter que la clé soit facilement retrouvée
- Eviter le stockage de la clé sur la machine. En effet, il est important de ne pas stocker "en dur" la clé qui permettra de déchiffrer les données de la victime. De plus, il est important de veiller à ce que la clé générée ne traine pas dans la mémoire vive d'une machine[6]

Sur base de ces critères, le fonctionnement de votre *ransomware* devra s'appuyer sur un chiffrement symétrique (AES en l'occurrence), plus rapide qu'un chiffrement asymétrique. De plus, le chiffrement symétrique permettra d'obtenir une clé unique liée au répertoire chiffré. Par contre, ce mode de fonctionnement nécessitera de communiquer la clé à l'attaquant. Il faudra donc utiliser le réseau pour "sortir" la clé de l'organisation de votre victime. Dans les critères de dépassement de ce projet, on vous invitera à proposer des solutions pour s'assurer que cette clé soit communiquée en limitant le risque que les victimes puissent identifier celle-ci (trace dans le réseau, dans la mémoire vive).

[6] voir la fonction memset pour écraser un emplacement mémoire: https://koor.fr/C/cstring/memset.wp

5.2 OpenSSL et ransomlib

Les fichiers *ransomlib.c ransom.h* utilisent la librairie OpenSSL pour chiffrer et déchiffrer des données. L'algorithme de chiffrement symétrique utilisé est l'AES_265_CBC, en mode bloc, dont le schéma ci-dessous résume le fonctionnement:

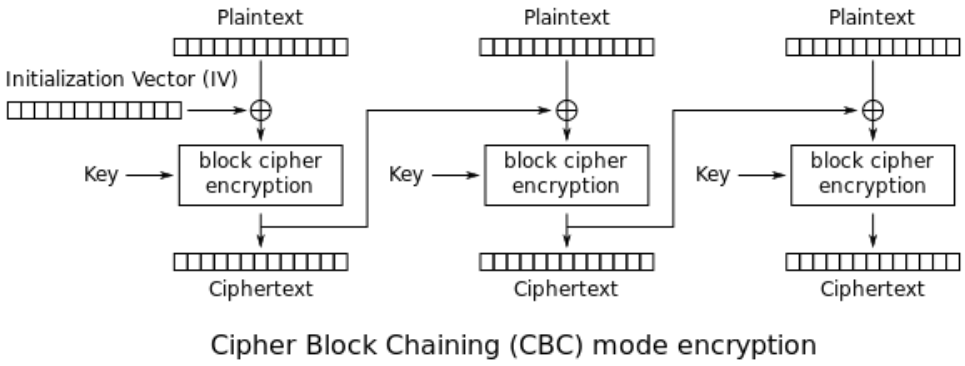

Figure 4: Chiffrement en mode bloc.

Le chiffrement d'un fichier à l'aide d'OpenSSL va donc passer par une initialisation (à l'aide du vecteur d'initialisation IV, qui permet de s'assurer de l'unicité du chiffrement) et puis ensuite par une boucle, qui s'occupera de découper le fichier en blocs (dont la taille est déterminée par la constante BUFSIZE). Chaque bloc sera alors chiffré sur base du précédent. Pour finir, la chaine chiffrée est recomposée à l'aide des différents blocs.

6 Evaluation

6.1 Acquis d'apprentissage

L'évaluation a pour objectif de valider les acquis d'apprentissage tels que décrits dans la fiche UE du cours:

- Développer une application C dans le contexte de la sécurité informatique
- Comprendre les enjeux de la gestion de la mémoire en C
- Argumenter les choix d'implémentation

6.2 Critères minimum de réussite

Votre système devra au minimum respecter les contraintes suivantes

- Programme qui compile et fonctionne sans erreurs.
- Respect des consignes de l'énoncé.

- Respect des bonnes pratiques vues durant les cours et laboratoires.
- Implémentation des fonctions définies dans le fichier *ransom.c*.
- Le programme est capable de chiffrer et de déchiffrer le contenu d'un répertoire de manière récursive.
- Utilisation des arguments pour renseigner le chemin du répertoire à chiffrer, le mode d'opération (chiffrer/déchiffrer) ainsi que la clé et le vecteur le cas échéant (argv).
- Communication de la clé et du vecteur d'initialisation via les sockets.
- Respect de la deadline.

6.3 Critères de dépassement

Une fois les critères de base rencontrés, libre aux étudiants de dépasser ces fonctionnalités et d'ajouter une touche personnelle à leur système. Ci-dessous une liste non exhaustive de critères de dépassement :

- Chiffrement RSA de la clé avant envoi par le réseau.
- Nettoyage de la mémoire pour supprimer toute trace d'une clé dans celle-ci.
- Gestion des fichiers volumineux qui pourrait ralentir inutilement le ransomware (ignorer les vidéos par exemple).
- Gestion du déchiffrement automatique.
- Gestion des versions [7].
- Originalité.
- ...

7 Avertissement

"A lutter avec les mêmes armes que ton ennemi, tu deviendras comme lui."[8]

Attention que le logiciel que vous allez développer peut être destructif dans le sens où un chiffrement mal géré peut rendre inutilisable vos données. Afin d'éviter tout problème, nous vous conseillons de:

- Travailler sur une machine virtuelle. Gardez un clone de celle-ci en cas de problème et sauvegardez régulièrement votre code (évitez de chiffrer votre code sans pouvoir le récupérer par exemple)

[7] https://github.com/
[8] Friedrich Nietzsche

- Ne pas lancer votre programme en mode root.

Notez également que la réalisation de votre *ransomware* s'inscrit dans une démarche pédagogique. Si vous le partagez ou l'hébergez sur une plateforme de type *Github*, veillez à toujours bien indiquer l'objectif poursuivi par le logiciel et rappelez le caractère didactique de son développement.

8 Modalités pratiques

- Le projet doit être réalisé par groupes de deux étudiants.
- Les étudiants devront être capables de défendre chaque partie du système.
- De plus, les étudiants seront interrogés de manière individuelle sur des questions théoriques liées au cours.
- Toute tentative de triche ou de plagiat sera sanctionnée d'un zéro.
- Les étudiants utiliseront deux machines virtuelles pour faire une démonstration de leur système. Les inscriptions se feront via Moodle.

9 Délivrable

Pour le 09/01/2022 à 23h59, les étudiants devront remettre un délivrable contenant les éléments suivants (un dépôt par groupe):

- Le code source de votre programme
- Référence éventuelle à un dépôt *Git*
- Une documentation aussi complète que possible expliquant le fonctionnement et les choix d'implémentation du système. Cette documentation fera office de rapport pour le projet. Le travail réalisé en autonomie doit être intégré de manière propre à votre rapport (sous la forme d'une analyse préliminaire du projet).

10 Le plagiat

Attention de toujours bien indiquer vos références concernant les parties de code dont vous n'êtes pas l'auteur[9].

11 Contact

En cas de problème/questions contactez en priorité le professeur en charge de votre laboratoire. Si nécessaire le professeur responsable de l'UE: adrien.voisin@henallux.be

[9] https://fr.wikipedia.org/wiki/Plagiat